Speeding up Parsing of Biological Context-Free Grammars

Daniel Fredouille and Christopher H. Bryant*

The Robert Gordon University, Aberdeen, UK.
http://www.comp.rgu.ac.uk/research/cig
Email: {chb|df}@comp.rgu.ac.uk

Abstract. Grammars have been shown to be a very useful way to model biological sequences families. As both the quantity of biological sequences and the complexity of the biological grammars increase, generic and efficient methods for parsing are needed. We consider two parsers for context-free grammars: depth-first top-down parser and chart parser; we analyse and compare them, both theoretically and empirically, with respect to biological data. The theoretical comparison is based on a common feature of biological grammars: the gap - a gap is an element of the grammars designed to match any subsequence of the parsed string. The empirical comparison is based on grammars and sequences used by the bioinformatics community. Our conclusions are that: (1) the chart parsing algorithm is significantly faster than the depth-first top-down algorithm, (2) designing special treatments in the algorithms for managing gaps is useful, and (3) the way the grammar encodes gaps has to be carefully chosen, when using parsers not optimised for managing gaps, to prevent important increases in running times.

1 Introduction

Among models used to represent sets of strings, *formal grammars* introduced by Chomsky [1] have been the subject of many studies. Searls [2] made the link between this formalism and structural phenomena found in biological sequences, showing the capabilities and limits of this formalism to represent biological sequences families. Numerous types of grammar formats are used for biological sequences analysis. Sub-regular patterns are used in PROSITE [3]; more complex patterns have been designed using Definite Clause Grammars (DCG) [4], String Variable Grammars (SVG) [5], PATSCAN patterns [6] or Basic Gene Grammars (BGG) [7]. For all but PATSCAN patterns, parsing is realised through algorithms available in the field of *context-free grammar parsing*: DCG and SVG with depth-first top-down parsing, and BGG with chart parsing. In this article, we focus on context-free grammars (CFG) parsers because they are generic[1],

* Contact author

[1] CFGs can represent a wide range of grammatical constructs, and special treatments can be easily added to the parsers to take into account constructs beyond context-free.

and are well studied[2]. However, to the best of our knowledge, they have never been analysed and compared when executed on biological data. Such a comparison has become important because there is a growing need for efficient parsers for biological grammars: the quantity of biological sequences is increasing daily and biological grammars are becomming more complex.

Our preliminary experiments made it clear to us that the respective performances of the considered parsers on biological data are closely related to a very common feature of biological grammars: the notion of *gap*. A gap is a rule designed to match any subsequence of the parsed string. This article compares the algorithms both theoretically (for grammars containing gaps) and empirically on biological strings and grammars. Our conclusions are: (1) the chart parser (CP) possesses a significant advantage in terms of running time over depth-first top-down parser (DFTDP), (2) designing special treatments for gaps in these parsers is useful, and (3), when using parsers not optimised for gaps, the gap rule design has to be carefully chosen to prevent important increases in running times.

After presenting the definitions (Section 2), we analyse the complexity of the DFTDP and the CP with respect to gaps (Sections 3 and 4). This analysis is then empirically validated on biological data (Section 5). The appendix contains the proofs (or sometimes hint of the proofs due to space restrictions) for the properties of the paper.

2 Definitions

Words, languages and grammars: For any finite set $\mathcal{S}$, we denote by $|\mathcal{S}|$ its cardinality and by $\mathcal{S}^*$ its free monoid, i.e., the set of all sequences made by concatenating 0 or more symbols of $\mathcal{S}$. An element of $\mathcal{S}^*$ is called a *word* over $\mathcal{S}$. The length of a word w is denoted by $|w|$, the word of length 0 or *empty word* is denoted by ϵ. Any set of words over $\mathcal{S}$ (i.e., any subset of $\mathcal{S}^*$) is called a *language* over $\mathcal{S}$. Languages can be represented by formal systems called *grammars*. The grammars we consider in this paper are *Context-Free*.

Definition 1. *A* Context-Free Grammar *(*CFG*) is a tuple* $\mathcal{G} = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ *where* $\mathcal{T}$ *and* $\mathcal{N}$ *are finite sets of symbols called respectively* terminals *and* nonterminals. $\mathcal{T}$ *and* $\mathcal{N}$ *are disjoint and their union is denoted by* $\mathcal{V} = \mathcal{T} \cup \mathcal{N}$. S *is a nonterminal called the* start symbol *and* $\mathcal{R}$ *is the set of* production rules*; each rule in* $\mathcal{R}$ *is of the form* $A \twoheadrightarrow \alpha$ *where* $A \in \mathcal{N}$ *is called the* left part *of the rule, and* $\alpha \in \mathcal{V}^*$. □

A CFG represents a language over $\mathcal{T}$ using the relation $\rightarrow$ defined by:

$$\forall u, v \in \mathcal{V}^* : \; u \rightarrow v \Leftrightarrow \exists \; (A \twoheadrightarrow \alpha) \in \mathcal{R}, \; \exists u_1, \; u_2 \in \mathcal{V}^*, \; u = u_1 A u_2, \; v = u_1 \alpha u_2.$$

We denote by $\rightarrow^*$ the transitive closure of $\rightarrow$; if $u \rightarrow^* v$, v is said to *derive* from u. The language $L(N)$ generated by a nonterminal N is the set of words over $\mathcal{T}$ derived from N, i.e., $L(N) \triangleq \{u \in \mathcal{T}^* : \; N \rightarrow^* u\}$. The language represented by

[2] Among others, in the field of natural language processing.

a grammar $\mathcal{G} = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ is $L(\mathcal{G}) \triangleq L(S)$.

Gaps rules: An *unlimited gap* is a nonterminal G such that $L(G) = \mathcal{T}^*$. For fixed values $lo, up \in \mathbb{N}$, a *limited gap* is a nonterminal G such that $L(G) = \{u \in \mathcal{T}^* : \ lo \leq |u| \leq up\}$. Numbers lo and up are respectively the *lower* and *upper bounds* of the limited gap, the value $ra = up - lo$ is its *range*.

Gaps can be implemented using different sets of grammar rules. To define these rules we use the following notation: N^i denotes nonterminal N repeated i times (e.g., $N^2 = NN$, and $N^0 = \epsilon$) and X is a nonterminal with rules $\{X \twoheadrightarrow t : \ t \in \mathcal{T}\}$. We study the following unlimited gap implementations: the *left-recursive* gap G_l with rules $\{G_l \twoheadrightarrow \epsilon, G_l \twoheadrightarrow G_l \ X\}$, and the *right-recursive* gap G_r with rules $\{G_r \twoheadrightarrow \epsilon, G_r \twoheadrightarrow X \ G_r\}$. We study the following limited gap implementations, with bounds lo and up fixed, and range $ra = up - lo$: the *linear* gap $G_{1,lo,up}$, with rules $\{G_{1,lo,up} \twoheadrightarrow X^{lo} \ G'_{1,ra}, \ G'_{1,ra} \twoheadrightarrow X_e^{ra}, \ X_e \twoheadrightarrow X, \ X_e \twoheadrightarrow \epsilon\}$, and the *quadratic* gap $G_{2,lo,up}$, with rules $\{G_{2,lo,up} \twoheadrightarrow X^{lo} \ G'_{2,ra}\} \cup \{G'_{2,ra} \twoheadrightarrow X^i : \ i \in [0, ra]\}$. The linear and quadratic gaps are so called because they respectively need, to represent the gap, a linear or a quadratic number of symbols in function of ra. It can be checked that these implementations respect the gap definitions, i.e.: $L(G_l) = L(G_r) = \mathcal{T}^*$ and $L(G_{1,lo,up}) = L(G_{2,lo,up}) = \{u \in \mathcal{T}^* : \ lo \leq |u| \leq up\}$. We denote by $Gaps$ the set of nonterminals we use to implement gaps, i.e., $Gaps \triangleq \{G_l, G_r, X, X_e\} \cup \{G_{1,lo,up}, G_{2,lo,up}, G'_{1,ra}, G'_{2,ra} : \ lo, up, ra \in \mathbb{N}, lo \leq up\}$.

Other notations: In the remainder of this paper we consider that: a grammar $\mathcal{G} = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ is given, with $\mathcal{V} = \mathcal{T} \cup \mathcal{N}$; that lower case letters $u, v, w, \ldots$ denote words over $\mathcal{T}$ with w being the word to parse by the grammar; that greek letters $\alpha, \beta, \ldots$ denote words over $\mathcal{V}$ and that lower case letters $i, j, \ldots$ denote natural numbers.

For a word u, $u[i{:}j]$ (with $i, j \in [0, |u|], \ i \leq j$) denotes the subword of u starting at position i and ending at position j excluded (e.g. $abcd[0{:}1] = a$, $abcd[2{:}4] = cd$, $abcd[1{:}1] = \epsilon$). $u[i]$, $u[i{:}]$ and $u[{:}i]$ are respectively shortcuts for $u[i{:}i{+}1]$, $u[i{:}|u|]$ and $u[0{:}i]$.

3 Parsing gaps using depth-first top-down parsing

In this section we consider the study of the depth-first top-down parser (DFTDP) with respect to gaps. Subsection 3.1 describes the principle of the DFTDP, then, in Subsections 3.2 and 3.3 we study respectively the effect of the implementation of gap rules on the DFTDP and a way of speeding it up when extra information is available on which rules represent gaps.

3.1 The depth-first top-down parser

The DFTDP can parse words in any CFG which is not left-recursive, i.e. such that $\forall N \in \mathcal{N} : \ \neg(N \rightarrow \alpha \rightarrow^* N\beta)$. Its worst complexity is non polynomial ($O(|w|^{|\mathcal{G}|})$), where $|\mathcal{G}|$ is the sum of the length of all rules in $\mathcal{G}$. However, this

Algorithm 1 Depth-first top-down parser, the initial call is DFTDP(w,S) with boolean *accepted* set to $false$.

1: **Function** DFTDP(u, α)
2: **if** $\alpha = \epsilon$ **then**{ **if** $u = \epsilon$ **then** $\{accepted \leftarrow true$; Stop algorithm$\}\}$
3: **else if** $\alpha[0] \in \mathcal{N}$ **then**{**for all** $\alpha[0] \rightarrowtriangle A_1 \dots A_n \in \mathcal{R}$ **do** DFTDP$(u, A_1 \dots A_n \alpha[1:])\}$
4: **else if** $\alpha[0] \in \mathcal{T}$ **then**{**if** $u \neq \epsilon$ **and** $u[0] = \alpha[0]$ **then** DFTDP$(u[1:], \alpha[1:])\}$

parser possesses the advantages of being very easy to implement, to need only $O(|\mathcal{G}| \times |w|)$ memory, and is considered to be fast for most grammars in practice.

The DFTDP uses the $\rightarrow$ relation to explore the space of all possible derivations of S and tries to find one equal to the input word w. This space is explored in a depth-first manner, and is pruned as soon as an incompatibility between the input word and the derivation currently considered is found. It can be described in a simplified manner by Algorithm 1. The word w is accepted by Algorithm 1 iff boolean *accepted* is true after execution.

3.2 Complexity of parsing gaps with DFTDP

The worst case execution time for the DFTDP is when the whole space of derivations has to be explored in order to reject a word. As we focus on gaps, we consider this worst case arising from the presence of a gap.

Concerning unlimited gaps, the DFTDP does not work on left-recursive rules implying that the right-recursive gap implementation has to be used. Property 1 shows that the DFTDP has a reasonable linear behaviour in this case.

Property 1. For a call DFTDP$(u, G_r a)$, with $u[|u|-1] \neq a$, the number of recursive DFTDP calls is linear in $|u|$.

Concerning limited gaps, from Property 2 we can expect the quadratic implementation to be far more efficient than the linear one.

Property 2. For a call DFTDP$(u, G_{1,lo,up})$ (resp. DFTDP$(u, G_{2,lo,up})$), with $|u| > up$, the number of recursive DFTDP calls is linear in lo and non polynomial (resp. quadratic) in $ra = up - lo$.

Conditions $u[|u| - 1] \neq a$ and $|u| > up$, respectively in Property 1 and 2, ensure the worst case complexity is reached by forcing the parsed word u to be rejected.

3.3 Adapting the algorithm

We have seen in the previous subsection which rules are adequate to represent gaps when using the DFTDP. In this subsection, we consider the option of implementing an algorithm based on the DFTDP, but adapted to gap rules. The proposed adaptation consists in changing line 3 of Algorithm 1 to obtain a special treatment of gap rules as shown by Algorithm 2. Property 3 shows that this modification deals with any kind of gaps in linear time, compared to quadratic or exponential behaviour given by Property 2.

Algorithm 2 Line 3 of Algorithm 1 with gap optimisation.

```
1: else if α[0] ∈ 𝒩 then
2:     if α[0] is a gap then if α[0] is limited then let lo and up be its bounds
3:                           else let lo ← 0 and up ← ∞
4:                           for all i ∈ [lo, min(|u|, up)] do DFTDP(u[i:], α[1:])
5:     else for all α[0] ⇸ A_1 ... A_n ∈ ℛ do DFTDP(u, A_1 ... A_n α[1:])
```

Property 3. We denote by DFTDP′ Algorithm 1 with modification of Algorithm 2, and by G a gap nonterminal with bounds lo and up (with $lo = 0$ and $up = \infty$ if the gap is unlimited). The call DFTDP′(u, Gb), with $b \in \mathcal{T}$ and $u[|u|-1] \neq b$ generates a number of recursive calls linear in $min(|u|, up) - lo$.

4 Parsing gaps using chart parsing

We study in this section the behaviour of the chart parser (CP) when it encounters gaps. Subsection 4.1 describes the CP, Subsection 4.2 examines the effect of the gap implementation on it and Subsection 4.3 provides a way of speeding it up when it is provided with extra information on which rules represent gaps.

4.1 Chart-parsing

The CP can parse any CFG (most CFG parsers are limited to subsets of CFG). Its worst complexity is polynomial, $O(|w|^3)$, and it has the advantage of representing all the alternative ways of parsing a sequence (potentially an exponential number) into a structure of polynomial size.

We will not detail the whole algorithm, only its main abstract data types and principles. For details on the algorithm, the interested reader can consult [8, 9]. The basic component of chart parsing is the *item*, this is a structure of the form : $R \twoheadrightarrow \alpha \bullet \beta @ i, j$. Such an item means that the α part of rule $R \twoheadrightarrow \alpha\beta$ can derive $w[i, j]$, more formally:

$$\alpha \rightarrow^* w[i{:}j] \tag{1}$$

Symbols $\bullet$ and @ are separators between respectively α and β, and between β and the couple i, j. R is called the *left part* of the item, and indice j is called the *ending position* of the item. Items are stored in a set called *itemset*. From Equation 1, w is accepted by the grammar iff *itemset* contains an item of the form $S \rightarrow \alpha \bullet @0, |w|$.

In its most studied implementation due to Earley [10, 11], the CP first inserts (in *itemset*) items which end at position 0, and then iteratively fills *itemset* with items ending at position j (j being incremented from 1 to $|w|$) . The CP stops after inserting items for $j = |w|$. In the Earley implementation, items also respect the following equation:

$$\exists \gamma, \delta \in \mathcal{V}^* \text{ such that: } S \rightarrow^* \gamma R \delta \text{ and } \gamma \rightarrow^* w[{:}i] \tag{2}$$

Equation 2 represents the fact that items with a left part that cannot be reached from S at position i of w are useless to check the acceptance of w, and therefore do not need to be stored. The only items which the Earley implementation stores in *itemset* are those which satisfy both Equations 1 and 2.

4.2 Complexity of parsing gaps with chart parsing

We will compare the gap implementations by assessing their complexity using the number of items the CP stores in *itemset*. The fewer items we need to store, the more efficient is the CP.

Property 4. Let $G = G_r$ when considering right-recursive gaps and $G = G_l$ when considering left-recursive ones. For $l \in [0, |w|]$, let $\mathcal{P}_l = \{k \in [0,l] : \exists R \twoheadrightarrow \alpha \bullet G\beta@j,k \in itemset,\ R \neq G\}$, and $p_1 = \min(\mathcal{P}_{|w|} \cup \{|w|+1\})$. Suppose $S \neq G$, the number of items in *itemset* with G as left part is S'_r for right recursive gaps, and S'_l for left-recursive gaps with: $S'_r = S'_l = 0$ iff $p_1 > |w|$; $S'_r = (1 - 5p_1/2 + p_1^2/2) + |w|(2.5 - p_1) + |w|^2/2$ and $S'_l \leq (2|w|+1)|\mathcal{P}_{|w|}| \leq (2|w|^2 + |w|)|$ iff $p_1 \leq |w|$.

From Property 4, for right-recursive and left-recursive gaps, *itemset* can contain a number (resp. S'_r and S'_l) of items in the worst case quadratic in the size of the parsed word. However, in practice, $|\mathcal{P}_{|w|}|$ representing the number of positions where a gap is started in w, it can be supposed small compared to $|w|$. Moreover, biological grammars often model a pattern to be found somewhere in the parsed word, implying the use of unlimited gap before the pattern, and in this case $p_1 = 0$. Considering these points, the following approximations can be made: $S'_r \simeq \frac{5|w|}{2} + \frac{|w|^2}{2}$ and $S'_l \simeq 2 * K * |w|$ where K is small, showing that right-recursive gaps imply a better behaviour of the CP than left-recursive ones.

Property 5. For $l \in [0, |w|]$, let $\mathcal{I}_1(l, ra)$ (resp. $\mathcal{I}_2(l, ra)$) be the set of items in *itemset* having l as ending position and with $G'_{1,ra}$ or X_e (resp. $G'_{2,ra}$) as left part. We have: $|\mathcal{I}_1(l, ra)| > |\mathcal{I}_2(l, ra)| + 2ra$

From Property 5, we can see that we will always have more items in *itemset* when using linear limited gaps instead of quadratic limited gaps, the difference being proportional to the $ra = up - lo$ value of the gap for items ending at position l. Considering all items, the difference can therefore be quite large (in $O(|w| * ra)$) implying that the use of quadratic gaps is recommended for chart parsing.

4.3 Adapting the algorithm

Some modifications of the CP have already been proposed to speed it up when encountering particular rules: [10] propose optimisations linked with rules of the form $A \twoheadrightarrow \epsilon$, and in [7] an optimisation for gap rules is briefly explained. We adapt this last one for the Earley implementation changing the algorithm the following way: (a) Items of the form $A \twoheadrightarrow \alpha \bullet \beta@i,j$ with $A \in Gaps$ are not

introduced in *itemset*. (b) Each time an item of the form $R \twoheadrightarrow \alpha \bullet G\beta@i,j$, with G a gap nonterminal, is inserted into *itemset*, we store $R \twoheadrightarrow \alpha G \bullet \beta@i,j$ into a set called *gapset*. (c) Before the algorithm fills in *itemset* with items ending at position k, we insert items $R \twoheadrightarrow \alpha G \bullet \beta@i,k$ iff item $R \twoheadrightarrow \alpha G \bullet \beta@i,j$ is present in *gapset* and $k \in [lo+j, up+j]$, where lo and up are the bounds of the gap, or 0, ∞ if the gap is unlimited.

With these modifications, no more item with $G \in Gaps$ as left part is inserted in *itemset*, but some items are inserted in *gapset*. The size of *gapset* is in the worst case of the order of $O(|w|^2)$ like the worst case given in Subsection 4.2. In practice, a quadratic behaviour is in fact very unlikely to happen since, as we already argued in Subsection 4.2, the number of position where a gap can start should be small for non artificial data (i.e., the number of possible k in items of the form $R \twoheadrightarrow \alpha \bullet G\beta@i,k$ is small).

5 Experimental comparison

The material of the experiments is available at `http://www.comp.rgu.ac.uk/staff/chb/research/data_sets/cpm05/README.html`. Running times have been obtained on a SunBlade 2500 (under SunOS 5.8).

5.1 Parsing protein grammars and sequences

Data: The first experimental comparison uses patterns of the PROSITE[3] database [3] as a source of grammars, and the UNIPROT[4] database [12] as source of sequences. We chose the PROSITE (resp. UNIPROT) database because it contains the largest collection of hand validated protein patterns (resp. protein sequences) in the world. PROSITE patterns can be seen as grammars with low expression power. (They represent a subset of regular languages). Their wide use by the bioinformatics community shows that they can be considered as potentially pertinent subparts of more complex biological grammars. Thus, their efficient parsing is a prerequisite to the efficient parsing of more complex grammars.
Experimental setting: For each one of a random sample of 500 PROSITE patterns, we sampled 1000 sequences from UNIPROT, and parsed these sequences with a grammar equivalent to the pattern. For each pattern-sequence couple, the parsing time has been stored.
Results: Table 1 gives the mean parsing times for pattern-sequence couples with the different parsers and gaps implementation. A particular event explained later made us separate one of the patterns from the others in the last column. The means are taken over the 499 reminding patterns and the 1000 sequences parsed in each pattern. Figure 1 presents these results graphically as a function of the length of the parsed word.

Considering non optimised versions of the parsers, the CP with left-recursive gaps and the DFTDP with right-recursive gaps have similar running times. For

[3] `http://www.expasy.ch/prosite/`
[4] `http://www.expasy.uniprot.org/`

Experiment Algo.	Gaps	Mean time	Standard deviation	PDOC00354 mean time
CP	G_l, G_1	$4.37ms$	$81.4ms$	$32.0ms$
CP	G_l, G_2	$4.36ms$	$81.4ms$	$16.4ms$
CP optimised		$1.54ms$	$62.4ms$	$1.20ms$
CP	G_r, G_1	$1.88s$	$4.82s$	$32.0ms$
CP	G_r, G_2	$1.88s$	$4.81s$	$16.5ms$
DFTDP	G_r, G_1	$4.35ms$	$83.0ms$	$> 86.4s^*$
DFTDP	G_r, G_2	$4.31ms$	$82.4ms$	$53.2ms$
DFTDP optimised		$0.63ms$	$49.8ms$	$0.350ms$

Table 1. Mean execution times per sequence on 499 PROSITE patterns and for the particular pattern PDOC00354.

$^*86.4s = 24h * 60mn * 60s/1000$ sequences

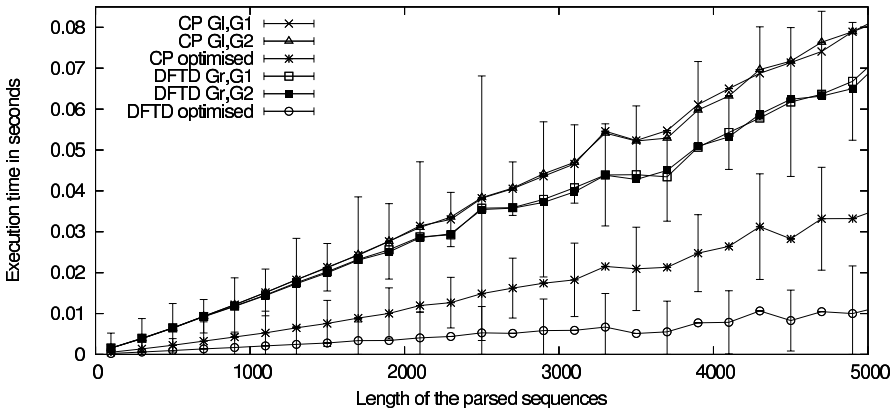

Fig. 1. Parsing time as a function of sequence length: each point is the mean parsing time for sequences with length ± 100 around the value of the x-axis. Only a subset of the standard deviations bars were plotted to keep the graph readable. Times for sequences lengths greater than 5000 were removed because less than 20 sequences were available for each mean. The results of the CP with G_r gaps were removed due to their large running time.

the CP, the unlimited gap implementation has a strong affect on running time, confirming the recommendation of Section 4.3 to use left-recursive gaps. For both parsers, important improvments have been obtained by using the modifications described in Section 3.3 and 4.3 (speeding up the parsers by a factor larger than 6 for the DFTDP and 2 for the CP).

For all but one of the patterns, the experiments do not show any significative difference between the linear and the quadratic gap implementations. The exception is the pattern PDOC00354: `<x(10,115)[DENF][ST][LIVMF][LIVSTEQ]Vx[AGP][STANEQPK]`. This pattern comprises a limited gap with range 105, followed by alternative choices for each position. The differences between this pattern and other PROSITE patterns are the position of the gap (at the beginning of the pattern) and its large range. This led to an important impact on the running time depending on the limited gap implementation. The DFTDP with linear gap had to be stopped on this pattern after 24 hours, showing the exponential blow-up predicted in Section 3.2. As a consequence, mean running times on this pattern

have been separated from the 499 remaining in the last column of Table 1. From this column we see that, as predicted in Sections 3.2 and 4.2, using the quadratic gap implementation leads to better running times for all parsers, despite the fact that the grammar is described with more symbols.

5.2 Parsing DNA grammars and sequences

This subsection compares parsing times for DNA grammars. These grammars represent more elaborate concepts than those of PROSITE.

Data: Different formalisms exist to represent patterns in DNA or RNA sequences, the one proposed by PATSCAN [6] has been widely used by the DNA/RNA modeling community. A collection of such patterns can be found on the UTRSITE[5] [13] which gives patterns modelling untranslated 3' and 5' regions of eukariotic mRNAs. This database is used to evaluate our approach for different reasons: (1) it is one of the rare places where a collection of patterns can be found, (2) the high degree of variation in the patterns gives rise to a wide variety of grammars providing a thorough test of the parsers, and (3) the patterns can generally be translated without too much difficulty into context-free grammars. The parsed sequences have been taken from UTRDB[6], the sequence database of UTRs associated with UTRSITE.

Experimental setting: PATSCAN patterns of UTRSITE were translated into a CFG [7]. Then, we randomly sampled 5000 sequences from the UTRDB database[8]. Finally we parsed the 5000 sequences in each CFG using those parsers which had a reasonable execution time on PROSITE patterns: DFTDP optimised, DFTDP with quadratic limited gaps, CP optimised, CP with left-recursive gaps and with either linear or quadratic limited gaps.

Results: Execution times for the different algorithms are given in Table 2.

The DFTDP had to be stopped for several grammars after running more than 24h to parse the 5000 sequences. A possible explanation of this is that there was an exponential blow-up for this parser due to features of PATSCAN patterns that were not present in PROSITE ones. This shows that the DFTDP is not adequate to parse complex biological grammars (even if the results of Subsection 5.1 on simple grammars were very good).

The optimised version of the DFTDP was even slower than the one using G_r and $G_{2,lo,up}$ gaps. A likely explanation is based on the number $\mathcal{Z}$ of executions of line 2 of algorithm 2. A blow-up implies $\mathcal{Z}$ to be exponential in the pattern size, if the encountered nonterminals at line 2 are not often gaps, this line brings

[5] `http://www2.ba.itb.cnr.it/UTRSite/`

[6] `http://www.ba.itb.cnr.it/srs7bin/cgi-bin/wgetz?-page+top`

[7] An exact translation was not possible in all cases, so we sometimes modified the patterns to obtain a slightly more general form that could be translated. We also removed some patterns that were "too simple", for example if the pattern was just a subword to be found in the sequence. Also, as we do not study here parsing with error-correcting, error limits were removed.

[8] Because some of the sequences in the database are very short (some are only one nucleic acid), we considered only sequences of length larger than 100.

Experiment		Mean	Standard
Algo.	Gaps	time	deviation
CP	G_l, G_1	$40.3ms$	$423ms$
CP	G_l, G_2	$40.7ms$	$429ms$
CP	optimised	$27.8ms$	$321ms$
DFTDP	G_r, G_2 *	$832s$	$2.63s$
DFTDP	optimised *	$940s$	$2.84s$

Table 2. Mean execution times per sequence on the 20 patterns of UTRDB

For experiments with the * symbol, 4 executions were stopped after running 24 hours.

few speed-up while spending an amount of ressources proportional to $\mathcal{Z}$ (i.e., exponential) to be evaluated.

The CP, on the other hand, showed a very stable behaviour: even if it is slower than the DFTDP for simple grammars, it seems more suitable for complex ones. We also see that the optimisations we added to the CP can be useful but are not needed if a grammar contains well designed gap rules.

6 Conclusion

We have studied two context-free grammar parsing algorithms. We have shown that for both the original and optimised versions of parsers, the CP is faster with respect to experimental data than the DFTDP. Indeed, even if the DFTDP can run much faster than the chart parser on simple biological grammars, its complexity increases drastically when provided with more complex grammars.

For algorithms not optimized to manage gaps, both our theoretical study of complexity, and our empirical study conclude that rules representing gaps have to be carefully choosen such as to obtain reasonable parsing times. On the other hand, when optimised algorithms can be implemented, an interesting speed-up can be achieved: between a factor of $\frac{4}{3}$ to 6 depending on both the considered algorithms and data.

References

1. Chomsky, N.: Three models for the description of language. IRE Trans. on Information Theory **2** (1956)
2. Searls, D.B.: The linguistics of DNA. American Scientist **80** (1992) 579–591
3. Falquet, L. *et multi al.*.: Protein data bank. Nucleic Acid Research **30** (2002) 235–238
4. Pereira, F., Warren, D.H.D.: Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. Artificial Intelligence **13** (1980) 231–278
5. Searls, D.B.: String variable grammar: A logic grammar formalism for the biological language of DNA. Journal of logic Programming **12** (1993)
6. Dsouza, M., Larsen, N., Overbeek, R.: Searching for patterns in genomic data. Trends in Genetics **13** (1997) 497–498
7. Leung, S.w., Mellish, C., Robertson, D.: Basic Gene Grammars and DNA-ChartParser for language processing of Escherichia coli promoter DNA sequences. Bioinformatics **17** (2001) 226–236

8. Grune, D., Jacobs, C.J.: Parsing techniques – a practical guide. Ellis Horwood, Chichester, England (1990)
9. Gazdar, G., Mellish, C.: Natural Language Processing in Prolog. Addison Wesley (1989)
10. Aycock, J., Horspool, R.N.: Practical Earley parsing. The Computer Journal **45** (2002)
11. Jay, E.: An efficient context-free parsing algorithm. Commun. ACM **13** (1970) 94–102
12. Apweiler, R. *et multi al.*: UniProt: the universal protein knowledgebase. Nucl. Acids Res. **32** (2004) D115–119
13. Pesole, G., Liuni, S.: Internet resources for the functional analysis of 5' and 3' untranslated regions of eukaryotic mRNA. Trends in Genetics **15** (1999) 378

Proofs of properties

Properties on the DFTDP

For properties 1 and 2, we denote by $|\textsc{dftdp}(v,\alpha)|$ the number of DFTDP calls realised when executing the call $\textsc{dftdp}(v,\alpha)$.

Property 1 The call of $\textsc{dftdp}(u, G_r b)$, with $b \in \mathcal{T},\ u[|u|-1] \neq b$, generates a number of recursive DFTDP calls linear in $|u|$.

Proof. We consider the value of $|\textsc{dftdp}(u, G_r b)|$ and proceed by induction. The base case is: $|\textsc{dftdp}(\epsilon, G_r b)| = |\mathcal{T}| + 3$. Indeed:

$$\begin{aligned}|\textsc{dftdp}(\epsilon, G_r b)| &= 1 + |\textsc{dftdp}(\epsilon, XG_r b)| + |\textsc{dftdp}(\epsilon, b)| \\ &= 1 + |\textsc{dftdp}(\epsilon, XG_r b)| + 1 \\ &= 1 + 1 + \Sigma_{t\in\mathcal{T}}|\textsc{dftdp}(\epsilon, tG_r b)| + 1 = |\mathcal{T}| + 3\end{aligned}$$

The induction step with $a \in \mathcal{T}$ is: $|\textsc{dftdp}(au, G_r b)| = |\mathcal{T}|+2+|\textsc{dftdp}(u, G_r b)|+|\textsc{dftdp}(au, b)|$. Indeed:

$$\begin{aligned}|\textsc{dftdp}(au, G_r b)| &= 1 + |\textsc{dftdp}(au, XG_r b)| + |\textsc{dftdp}(au, b)| \\ &= 1 + (1 + \Sigma_{t\in|T|}|\textsc{dftdp}(au, tG_r b)|) + |\textsc{dftdp}(au, b)| \\ &= 1 + (1 + |\textsc{dftdp}(au, aG_r b)| + |\mathcal{T}| - 1) + |\textsc{dftdp}(au, b)| \\ &= 1 + (1 + 1 + |\textsc{dftdp}(u, G_r b)| + |\mathcal{T}| - 1) + |\textsc{dftdp}(au, b)|\end{aligned}$$

We have $|\textsc{dftdp}(au, b)| = 1$ if $a \neq b$ and $|\textsc{dftdp}(au, b)| = 1+|\textsc{dftdp}(u, \epsilon)| = 2$ if $a = b$. The induction step stays valid because the call $\textsc{dftdp}(\epsilon, \epsilon)$ will never be considered due to the condition $u[|u| - 1] \neq b$ (the algorithm will not stop because of the condition of line 2, implying that all considered calls will be executed). Therefore, we have the following recursive equation:

$$\begin{cases} |\textsc{dftdp}(\epsilon, G_r b)| = |\mathcal{T}| + 3 \\ |\textsc{dftdp}(au, G_r b)| = |\mathcal{T}| + 2 + |\textsc{dftdp}(u, G_r b)| + |\textsc{dftdp}(au, b)| \end{cases}$$

which admit as solution:

$|\textsc{dftdp}(u, G_r b)| = |\mathcal{T}| + 3 + \Sigma_{1\leq i\leq|u|}(|\mathcal{T}| + 2 + |\textsc{dftdp}(u, b)|)$
$= |\mathcal{T}| + 3 + |u|(|\mathcal{T}| + 2) + \Sigma_{1\leq i\leq|u|}|\textsc{dftdp}(u, b)|$
$= |\mathcal{T}| + 3 + |u|(|\mathcal{T}| + 2) + |u| + 1 + K$ where K is the number of letters equal to b in u
$= (|u| + 1)(|\mathcal{T}| + 3) + K.$ □

Property 2 The call $\text{DFTDP}(u, G_{1,lo,up})$ (resp. $\text{DFTDP}(u, G_{2,lo,up})$), with $|u| > ra$ generates a number of recursive DFTDP calls linear in lo and non-polynomial (resp. quadratic) in $ra = up - lo$.

Proof.
We first consider the calls $\text{DFTDP}(u, G_{1,lo,up})$ (resp. $\text{DFTDP}(u, G_{2,lo,up})$) generates a linear number of calls in function of lo, and then calls $\text{DFTDP}(u, G'_{1,ra})$ (resp. $\text{DFTDP}(u, G'_{2,ra})$), where the quadratic (resp. non-polynomial) complexity is observed. Depending on the gap implementation, let $G_{lo,up}$ denote $G_{1,lo,up}$ or $G_{2,lo,up}$, and G'_{ra} denote $G'_{1,ra}$ or $G'_{2,ra}$.

Call $\text{DFTDP}(u, G_{lo,up})$:
We have: $|\text{DFTDP}(u, G_{lo,up}|) = 1 + \text{DFTDP}(u, X^{lo}G'_{ra})$ (rule $G_{lo,up} \rightarrow X^{lo}G'_{ra}$, and line 3 of algorithm 1). We show by induction that: $\text{DFTDP}(u, X^{lo}G'_{ra}) = (|\mathcal{T}|+1) * lo + |\text{DFTDP}(u, X^{lo}G'_{ra})|$. The base case is: $|\text{DFTDP}(u, X^0 G'_{ra})| = |\text{DFTDP}(u, G'_{ra})|$.
For $lo > 0$, the induction step is:
$|\text{DFTDP}(au, X^{lo}G'_{ra})| = 1 + \Sigma_{t\in\mathcal{T}}|\text{DFTDP}(au, tX^{lo-1})|$
$= 1 + |\mathcal{T}| - 1 + |\text{DFTDP}(au, aX^{lo-1})| = |\mathcal{T}| + 1 + |\text{DFTDP}(u, X^{lo-1})|$
From the base case and the induction step, we have for $lo \geq 0$:
$|\text{DFTDP}(au, X^{lo}G'_{ra})| = |\text{DFTDP}(u, G'_{ra})| + \Sigma_{1\leq i\leq lo}|\mathcal{T}| + 1$
$= |\text{DFTDP}(u, G'_{ra})| + lo * (|\mathcal{T}| + 1)$.
We now consider separately the proof for the linear and quadratic gap implementation.

Call $\text{DFTDP}(u, G'_{1,ra})$:
We have $|\text{DFTDP}(u, G'_{1,ra})| = 1 + |\text{DFTDP}(u, X_e^{ra})|$ (rule $G'_{1,ra} \rightarrow X_e^r$, line 3 of Algorithm 1).

We consider the value of $|\text{DFTDP}(u, X_e^{ra})|$ and proceed by induction. The base case is: $|\text{DFTDP}(u, X_e^0)| = |\text{DFTDP}(u, \epsilon)| = 1$ due to line 2 of Algorithm 1.
For $ra > 0$, the induction step is:
$|\text{DFTDP}(au, X_e^{ra})| = 1 + |\text{DFTDP}(au, XX_e^{ra-1})| + |\text{DFTDP}(au, X_e^{ra-1})|$
$= 1 + (1 + \Sigma_{t\in\mathcal{T}}|\text{DFTDP}(au, tX_e^{ra-1})|) + |\text{DFTDP}(au, X_e^{ra-1})|$
$= 1 + (1 + |\text{DFTDP}(au, aX_e^{ra-1})| + |\mathcal{T}| - 1) + |\text{DFTDP}(au, X_e^{ra-1})|$
$= 1 + (1 + 1 + |\text{DFTDP}(u, X_e^{ra-1})| + |\mathcal{T}| - 1) + |\text{DFTDP}(au, X_e^{ra-1})|$
$= 2 + |\mathcal{T}| + |\text{DFTDP}(u, X_e^{ra-1})| + |\text{DFTDP}(au, X_e^{ra-1})|$
We have $|\text{DFTDP}(u, X_e^{ra-1})| = |\text{DFTDP}(au, X_e^{ra-1})|$ due to the condition $|u| > ra$ and therefore: $|\text{DFTDP}(au, X_e^{ra})| = 2 + |\mathcal{T}| + 2|\text{DFTDP}(u, X_e^{ra-1})|$
As $\forall i \geq 1$, $|\text{DFTDP}(u, X_e^{ra})| > f(|u|)$, where $f(0) = 1$ and $\forall i > 0$, $f(i) = 2f(i-1) = 2^i$, $|\text{DFTDP}(u, X_e^{ra})| > f(|u|)$ is not polynomial in i.

Call $\text{DFTDP}(u, G'_{2,ra})$:
We have $|\text{DFTDP}(u, G'_{2,ra}|) = 1 + \Sigma_{0\leq j\leq ra}|\text{DFTDP}(u, X^j)|$.

We consider the value of $|\text{DFTDP}(u, X^j)|$ $(\forall j \geq 0)$ and proceed by induction. The base case is: $|\text{DFTDP}(u, X^0)| = |\text{DFTDP}(u, \epsilon)| = 1$ due to line 2 of Algorithm 1. The induction step is for $j > 0$:
$|\text{DFTDP}(au, X^j)| = 1 + \Sigma_{t\in\mathcal{T}}|\text{DFTDP}(au, tX^{j-1})|$

$= 1 + |\text{DFTDP}(au, aX^{j-1})| + |\mathcal{T}| - 1 = 1 + 1 + |\text{DFTDP}(u, X^{j-1})| + |\mathcal{T}| - 1$
$= |\mathcal{T}| + 1 + |\text{DFTDP}(u, X^{j-1})|$
From the base case and the induction step, we have $\forall j \geq 0$:
$|\text{DFTDP}(au, X^j)| = 1 + \Sigma_{1 \leq i \leq j}(|\mathcal{T}| + 1) = 1 + j(|\mathcal{T}| + 1)$
And therefore:
$|\text{DFTDP}(u, G'_{2,ra})| = 1 + \Sigma_{0 \leq j \leq ra}|\text{DFTDP}(u, X^j)| = 1 + \Sigma_{0 \leq j \leq ra}(1 + j(|\mathcal{T}| + 1))$
$= 1 + (ra + 1) + (|\mathcal{T}| + 1) * \frac{ra(ra+1)}{2}$ □

Proof of property 3 We denotes by DFTDP′ the algorithm 1 with modification of algorithm 2, and by G a gap nonterminal with bounds lo and up (with $lo = 0$ and $up = \infty$ if the gap is unlimited). The call DFTDP′(u, Gb), with $b \in \mathcal{T}$ and $u[|u| - 1] \neq b$ generates a number of recursive calls linear in $min(|u|, up) - lo$.

Proof. We consider the value of $|\text{DFTDP}'(u, Gb)|$. From line 4 of algorithm 2, this imply recursive calls of the form $|\text{DFTDP}'(u[i:], b)|$ for $i \in [lo, min(|u|, up)]$. Therefore:
$|\text{DFTDP}'(u, Gb)| = 1 + \Sigma_{i \in [lo, min(|u|, up)]}|\text{DFTDP}'(u[i:], b)|$
DFTDP′$(u[i:], b)$ calls imply a new call (and only one) iff $u[0] = b$, so if we denote by K the number of b in $u[lo : min(|u|, up) + 1]$, we have:
$|\text{DFTDP}'(u, Gb)| = 1 + (min(|u|, up) - lo) + K.$ □

Properties on the CP

When evaluating the performances of the CP, we will not count items in *itemset* with X as left part neither those of the form $N \twoheadrightarrow \epsilon \bullet \epsilon @ i, j$. Indeed counting them would not realise a fair comparison between the gap implementations since optimisations of the CP exist that suppress (the vast majority of) these items from the *itemset*[9].

For the proofs of this section, we will use the following notations: The symbol G will represent $G = G_r$ when considering right-recursive gaps, $G = G_l$ when considering left-recursive gaps, $G = G'_{1,ra}$ for linear limited gaps and $G = \{G'_{2,ra}\}$ for quadratic limited gaps. For $l \in [0, |w|]$, let $\mathcal{P}_l = \{k \in [0, l] : \exists R \twoheadrightarrow \alpha \bullet G\beta @ j, k \in itemset,\ R \neq G\}$, and $p_1 = \min(\mathcal{P}_{|w|} \cup \{|w| + 1\})$.

Intuitively, $\mathcal{P}_l$ is the set of positions in $w[0 : l+1]$ where an unlimited gap can be started, or where the range of a limited gap starts to influence the algorithm[10], and p_1 is the first of these positions for $w[0 : |w| + 1] = w$.

For these properties, we will also suppose that $S \notin Gaps$), indeed, the case $S \in Gaps$ is not useful in practice (such grammars are too simple to be used in practice), and taking them into account complexify the proofs.

[9] Optimisation known under the name of *look-ahead* implies that most items with X as left part do not have to be stored [8]; and an optimisation due to [10] implies that items of the form $N \twoheadrightarrow \epsilon \bullet \epsilon @ i, j$ do not have to be stored either.

[10] For limited gaps, as the rules $G_{1,lo,up} \twoheadrightarrow X^{lo} G'_{1,up-lo}$ and $G_{2,lo,up} \twoheadrightarrow X^{lo} G'_{2,up-lo}$ are similar, they will not be useful to characterize the difference between the implementations, this is why we focus on rules $G'_{1,ra}$ and $G'_{2,ra}$.

Proof of property 4 In this subsection, for $l \in [0, |w|]$, the set of items in *itemset* with G as left part and ending at position l are denoted $\mathcal{I}_r(l)$ for right-recursive gaps and $\mathcal{I}_l(l)$ for left-recursive gaps.

Property 4 Suppose $S \neq G$, the number of items in *itemset* with G as left part is S'_r for right recursive gaps, and S'_l for left-recursive gaps with:
$S'_r = S'_l = 0$ iff $p_1 > |w|$
$S'_r = (1 - 5p_1/2 + p_1^2/2) + |w|(2.5 - p_1) + |w|^2/2$ and $S'_l \leq (2|w| + 1)|\mathcal{P}_{|w|}| \leq (2|w|^2 + |w|)|$ iff $p_1 \leq |w|$.

Proof. Let $l \in [0, |w|]$, from Lemma 1, *itemset* contains all and only items with ending position equal to l and with G as left part of rule that are present in the set $\mathcal{I}_r(l)$ for right-recursive gaps and $\mathcal{I}_l(l)$ for left-recursive gaps. The size of $\mathcal{I}_r(l)$ (resp. $\mathcal{I}_l(l)$) can be easily deduced from Lemma 1, we denote it $S_r(l)$ (resp. $S_l(l)$), with:

$$S_r(l) = \begin{cases} 2 + l - p_1 \text{ if } p_1 < l \\ 1 + l - p_1 \text{ if } p_1 = l \\ 0 \text{ if } p_1 > l \end{cases} \quad \text{and} \quad S_l(l) = \begin{cases} 2 * |\mathcal{P}_l| \text{ if } l \notin \mathcal{P}_l \\ 2 * |\mathcal{P}_l| + 1 \text{ if } l \in \mathcal{P}_l \end{cases}$$

If $p_1 = |w| + 1$, we have trivially $S'_r = S'_l = 0$, otherwise ($p_1 \leq |w| + 1$) we obtain values for S_r and S'_l by summing $S_r(l)$ and $S_r(l)$ for all possible l:

$$\begin{aligned} S'_r &= \Sigma_{l \in [0,|w|]} S_r(l) = \Sigma_{l \in [p_1,|w|]}(2 + l - p_1) - 1 \\ &= (1 - 5p_1/2 + p_1^2/2) + |w|(2.5 - p_1) + |w|^2/2 \\ S'_l &= \Sigma_{l \in [0,|w|]} S_l(l) = |\mathcal{P}_{|w|}| + \Sigma_{l \in [0,|w|]}(2 * |\mathcal{P}_l|) \end{aligned}$$

$\Rightarrow S'_l \leq |\mathcal{P}_{|w|}| + 2 * |w| * |\mathcal{P}_{|w|}| \Rightarrow S'_l \leq |w| + 2 * |w|^2$ □

Lemma 1. *Suppose $S \neq G$, then:*

$$\begin{aligned} \mathcal{I}_r(l) = \quad & \{G_r \twoheadrightarrow \bullet X G_r @l, l : l \geq p_1\} && (r.1) \\ & \cup \{G_r \twoheadrightarrow X \bullet G_r @l\text{-}1, l : l > p_1\} && (r.2) \\ & \cup \{G_r \twoheadrightarrow X G_r \bullet @i, l : \ k \in \mathcal{P}_l, k \leq i < l\} && (r.3) \end{aligned}$$

and

$$\begin{aligned} \mathcal{I}_l(l) = \quad & \{G_l \twoheadrightarrow \bullet G_l X @l, l : \ l \in \mathcal{P}_l\} && (l.1) \\ & \cup \{G_l \twoheadrightarrow G_l \bullet X @k, l : \ k \in \mathcal{P}_l\} && (l.2) \\ & \cup \{G_l \twoheadrightarrow G_l X \bullet @k, l : \ k \in \mathcal{P}_l, k < l\} && (l.3) \end{aligned}$$

Proof. We use equations 1 and 2 to show respectively that $G_l \twoheadrightarrow \bullet G_l X @i, l \in itemset \wedge S \neq G_l$ is equivalent to $i = l \wedge l \in \mathcal{P}_l \wedge S \neq G_l$, and that $G_r \twoheadrightarrow \bullet X G_r @i, l \in itemset \wedge S \neq G_l$ is equivalent to $i = l \wedge l \geq p_1 \wedge S \neq G_r$.

From this first step, which corresponds to the content of line respectively l.1 and r.1, we can show similar properties for lines r.2, r.3 and l.2, l.3. Due to space restrictions the full proof is not included in the paper.

Proof of Property 5 We use the following definitions in this subsection: let $\mathcal{I}_1(l, ra)$ be the set of items with $G'_{1,ra}$ or X_e as left part of rule and l as ending position present in *itemset*. Similarly, let $\mathcal{I}_2(l, ra)$ be the set of items with $G'_{2,ra}$ as left part of rule and l as ending position present in *itemset*.

Property 5 $|\mathcal{I}_1(l, ra)| > |\mathcal{I}_2(l, ra)| + 2ra$.

Proof. Lemma 2 shows which items are present in each of the $\mathcal{I}_1(l, ra)$ and $\mathcal{I}_2(l, ra)$ sets. From Lemma 2 equations, we can see that the part 4.1 of $\mathcal{I}_1$ is always larger than $\mathcal{I}_2$, and that parts 4.2 and 4.3 of $\mathcal{I}_1$ have size ra, showing that $|\mathcal{I}_1(l, ra)| > |\mathcal{I}_2(l, ra)| + 2ra$. □

Lemma 2. *The $\mathcal{I}_1(l, ra)$ and the $\mathcal{I}_2(l, ra)$ contents respects the following equations:*

$$\begin{aligned}\mathcal{I}_1(l, ra) \supseteq & \{G'_{1,ra} \twoheadrightarrow X_e^{i+l-k} \bullet X_e^{ra-i-l+k)} @k, l \ : \ k \in \mathcal{P}_l, i \in [0, ra+k-l]\} && (4.1)\\ & \cup \{X_e \twoheadrightarrow \bullet X @l, l : \ \ k \in \mathcal{P}_l, \ l \in [k, k+ra[\} && (4.2)\\ & \cup \{X_e \twoheadrightarrow X \bullet @l-1, l : \ k \in \mathcal{P}_l, \ l \in]k, k+ra]\} && (4.3)\end{aligned}$$

And

$$\mathcal{I}_2(l, ra) = \{G \twoheadrightarrow X^{l-k} \bullet X^j @k, l \ : k \in \mathcal{P}_l, \ j \in [0, ra+k-l], \ l \neq k \ \vee \ j \neq 0\}.$$

Proof. For $\mathcal{I}_1(l, ra)$, the proof consists in first showing that $G'_{1,ra} \twoheadrightarrow X_e^j \bullet Xe^{ra-j} @k, l \in itemset \wedge S \neq G'_{1,ra}$ is equivalent to $k \in \mathcal{P}_l \wedge i = k - l + j \wedge 0 \leq i \leq k - l + ra$ by using equations 1 and 2. Then it can be shown from this first result and some more use of equations 1 and 2 that this implies the presence of at least the items of line 4.2 and 4.3 (other items are the ones comming from different possible values for ra). For $\mathcal{I}_2(l, ra)$, the proof is:

$$\begin{cases} G'_{2,ra} \twoheadrightarrow X^i \bullet X^j @k, l \in itemset \\ S \neq G'_{2,ra} \end{cases} \Leftrightarrow \exists \gamma, \delta \in \mathcal{V}^* \quad : \begin{cases} X^i \twoheadrightarrow^* w[k, l] \\ S \rightarrow^* \gamma G'_{2,ra} \delta \\ \gamma \twoheadrightarrow^* w[0{:}k] \\ 0 \leq i + j \leq ra \\ S \neq G'_{2,ra} \end{cases} \Leftrightarrow$$

$$\begin{array}{l} \exists m \in \mathbb{N}, \\ \exists R \twoheadrightarrow \alpha G'_{2,ra} \beta \in \mathcal{R}, \\ \exists \gamma', \delta' \in \mathcal{V}^* \end{array} \quad : \begin{cases} i = l - k \\ S \rightarrow^* \gamma' R \delta' \rightarrow \gamma' \alpha G'_{2,ra} \beta \delta' \\ \gamma' \twoheadrightarrow^* w[0{:}m] \\ \alpha \twoheadrightarrow^* w[m{:}k] \\ 0 \leq j + l - k \leq ra \end{cases} \Leftrightarrow$$

$$\begin{array}{l} \exists m \in \mathbb{N}, \\ \exists R \twoheadrightarrow \alpha G'_{2,ra} \beta \in \mathcal{R} \end{array} \quad : \begin{cases} i = l - k \\ R \twoheadrightarrow \alpha \bullet \beta @m, k \in itemset \\ 0 \leq j + l - k \leq ra \end{cases} \Leftrightarrow \begin{cases} i = l - k \\ k \in \mathcal{P}_l \\ k - l \leq j \leq ra + k - l \end{cases}$$

□